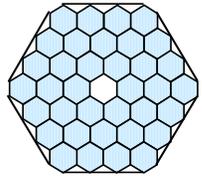


# Using Scripting Languages in Optical Interferometry

Leonard J. Reder, Thomas G. Lockhart,  
Kenneth C. Ko, Benjamin T. Smith  
August 28, 2002

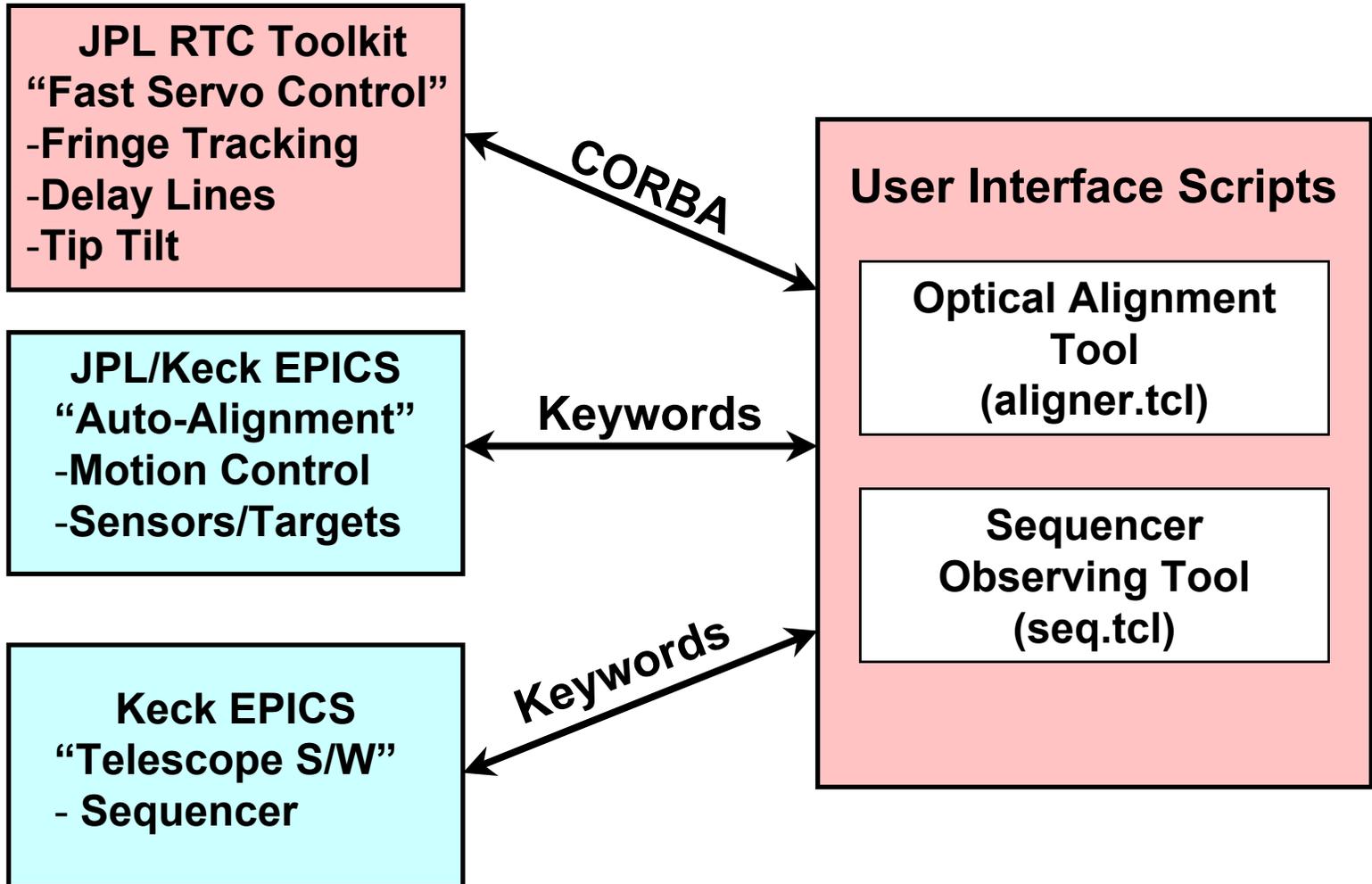
[Leonard.J.Reder@jpl.nasa.gov](mailto:Leonard.J.Reder@jpl.nasa.gov)

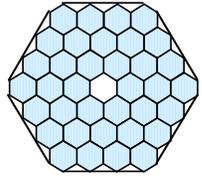
Real Time Interferometry Software Group  
Jet Propulsion Laboratory  
California Institute of Technology



# Keck Interferometer Software

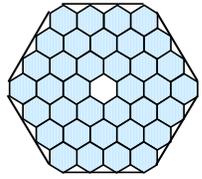
**KECK INTERFEROMETER  
PROJECT**



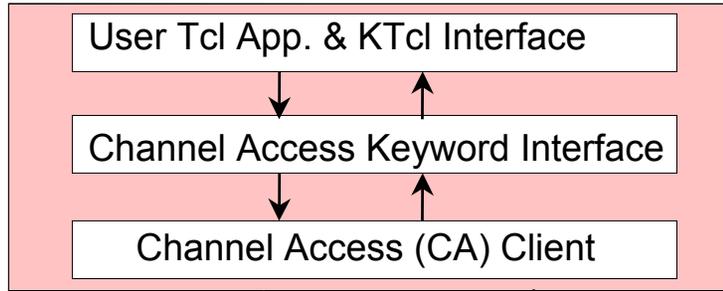


# What I will talk about

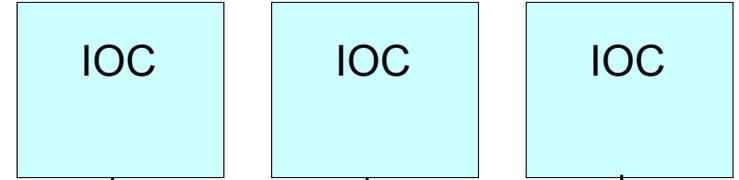
- Two client (Tcl/Tk) applications
  - EPICS/CA/ Keywords (KTL, KTcl)
  - Optical alignment control user interface (aligner.tcl)
    - Manual and automatic
  - Sequencer front end (seq.tcl)
    - History (EPICS based, then Rhapsody C++)
    - CORBA (Mico ORB and Combat)
- Detailed example of a simple CORBA client & server script
  - IDL
  - Tcl client
  - Python server



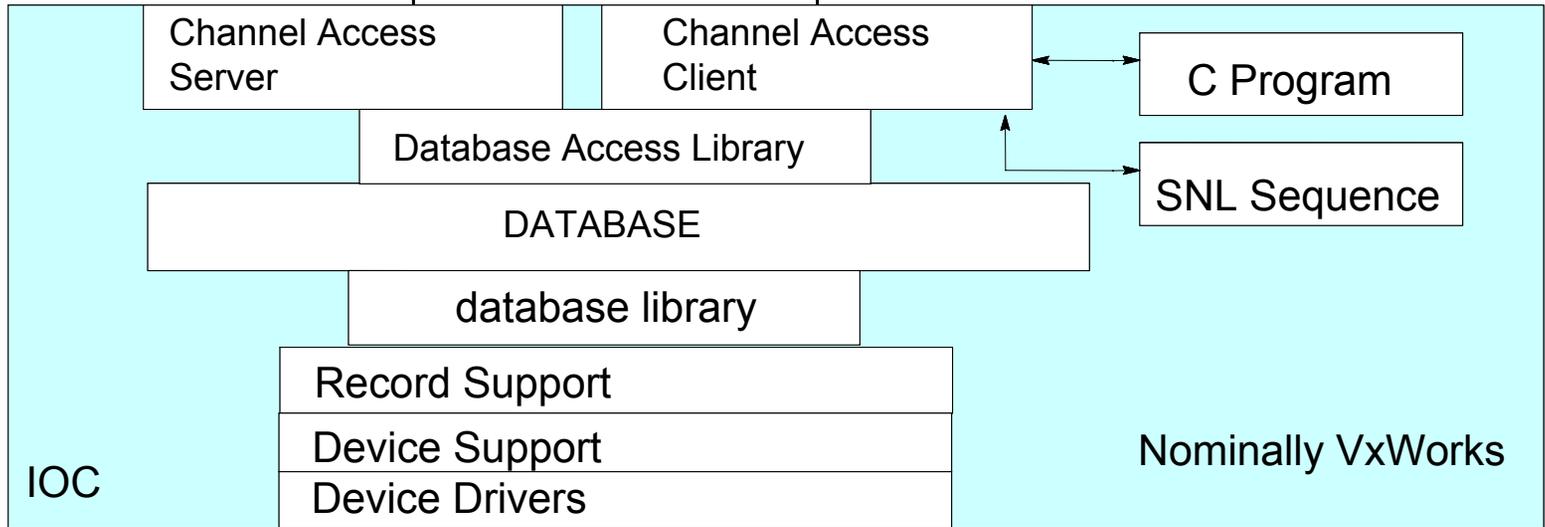
### WORKSTATION



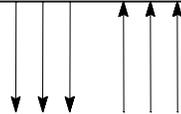
# EPICS Architecture

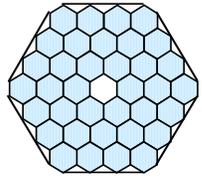


TCP/IP



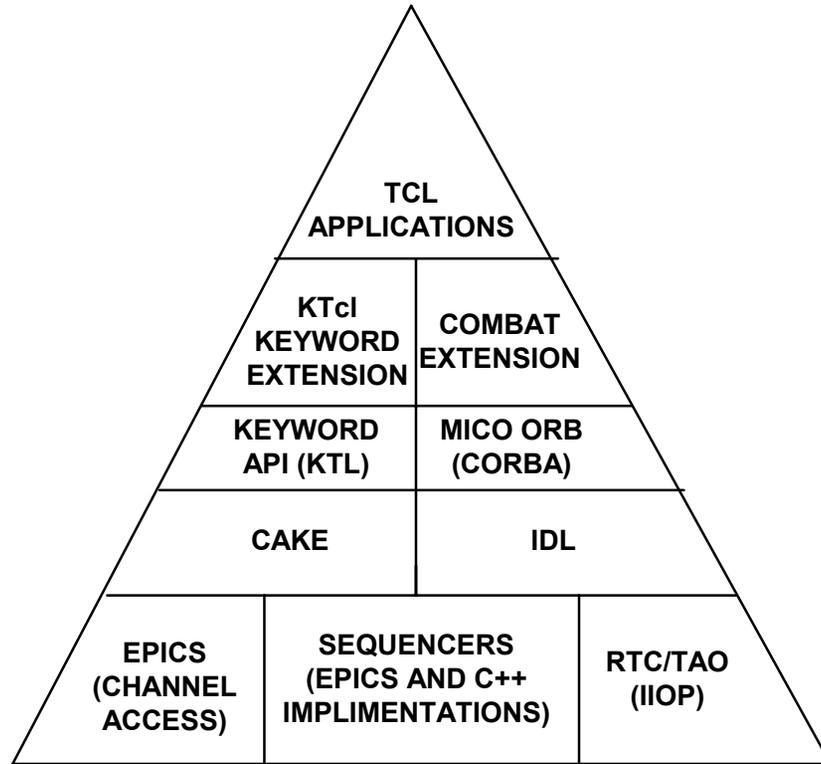
Instrumentation And Control Hardware

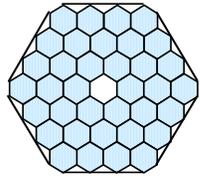




# Software Layers within TCL Applications

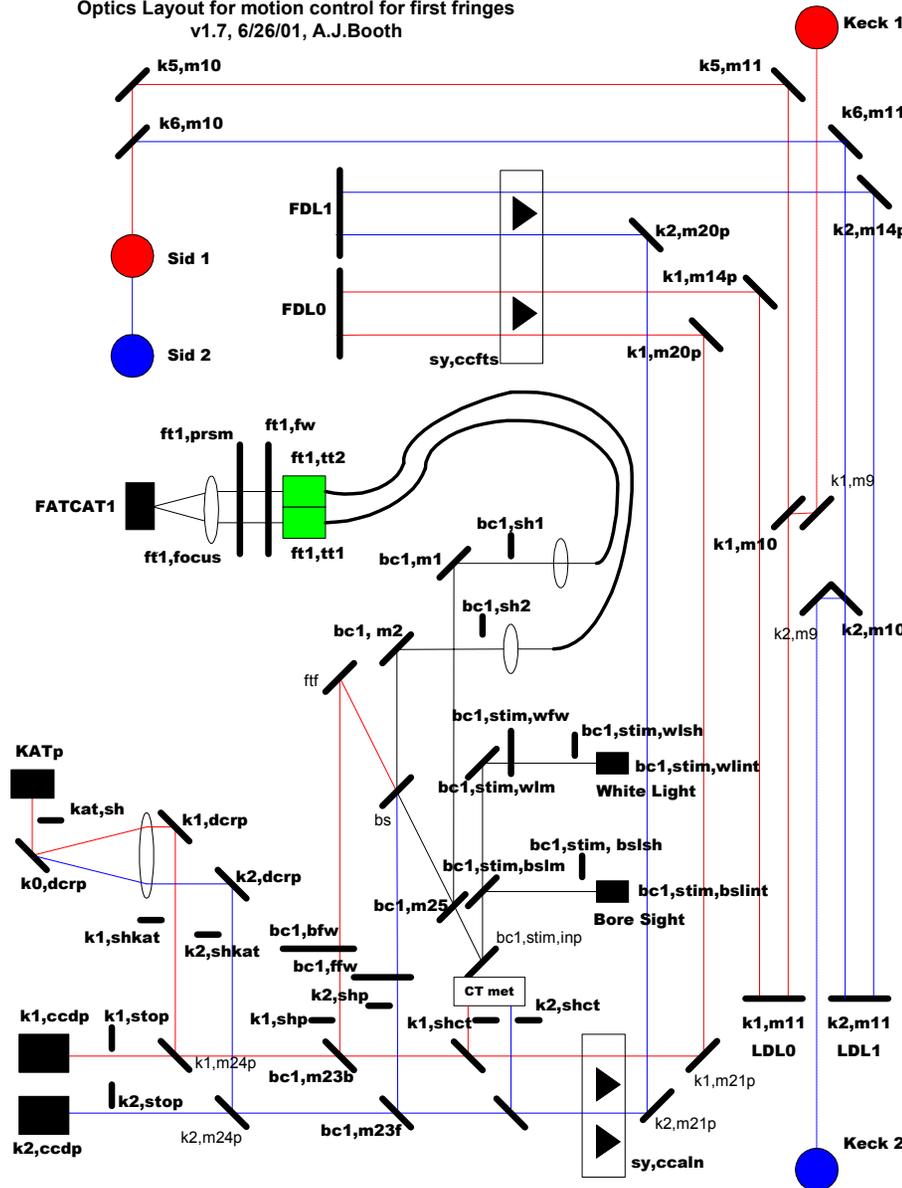
**KECK INTERFEROMETER  
PROJECT**





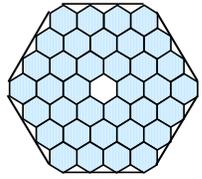
# Optics and Control Layout for first fringes

Optics Layout for motion control for first fringes  
v1.7, 6/26/01, A.J.Booth



- AA Sequencers:
  - 6 Beam Train
  - 2 Internal
- Sensors:
  - 2 C.T.
  - 4 Cameras
- Stimuli Targets:
  - White Light
  - Laser Boresite
  - 12 LEDs
- Actuators:
  - 20 Newport 850G
  - 26 Pico Motors
  - 16 Shutters
  - Other Misc.

KECK INTERFEROMETER  
PROJECT



# Automatic and Manual Optical Alignment User Interface TCL Application (aligner.tcl)

KECK INTERFEROMETER PROJECT

Alignment Position Status

AA Sequencer Monitors & Controls

AA Mirror Select

Text Status

Actuator Status

Sensor Select

Alignment Vector & AA Camera Image

Aligner Revision: 1.36 (Initial Version with btSeq connected)

Align	Sensor	Actuator	Stimulus
<b>Position Status</b>			
Ref. Bore-site:	⟨ 0.00, 0.00, 0 ⟩		
Align Point:	⟨0,0⟩		
Centroid:	⟨ 87.257, 91.654 ⟩		
<b>Sequencer Parameters:</b>			
BT Seq. Mirror:	K1.M6		
BT Seq. State:	CENTROIDING		
<b>Seq. Motor Transformations:</b>			
Rotation :	196.710000		
X Gain:	-35.430000		
Y Gain:	27.500000		

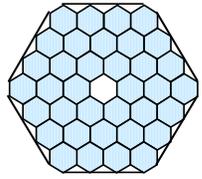
Units : Pixels

HALT STBY INIT ALIGN MOVE X: 100 Y: 100

BT Sequencer	Actuator (Status)	Sensor
K1.M6	K1.M6P (INPOS)	K1.CCDP
K1.M7P	K1.M7P (INPOS)	K2.CCDP
K1.M14P	K1.M14P (INPOS)	
K2.M6	K2.M6P (STBY)	

Updated K1.M6P.ERRORS: Loop open  
Updated K1.M6.ERRORSTR: OK  
Updated K1.M6.ERRORSTR: OK  
Canvas Update: XGEN = 87.2573, YGEN = 91.6543  
Canvas Update: XGEN = 87.2573, YGEN = 91.6543

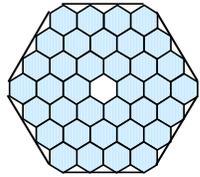




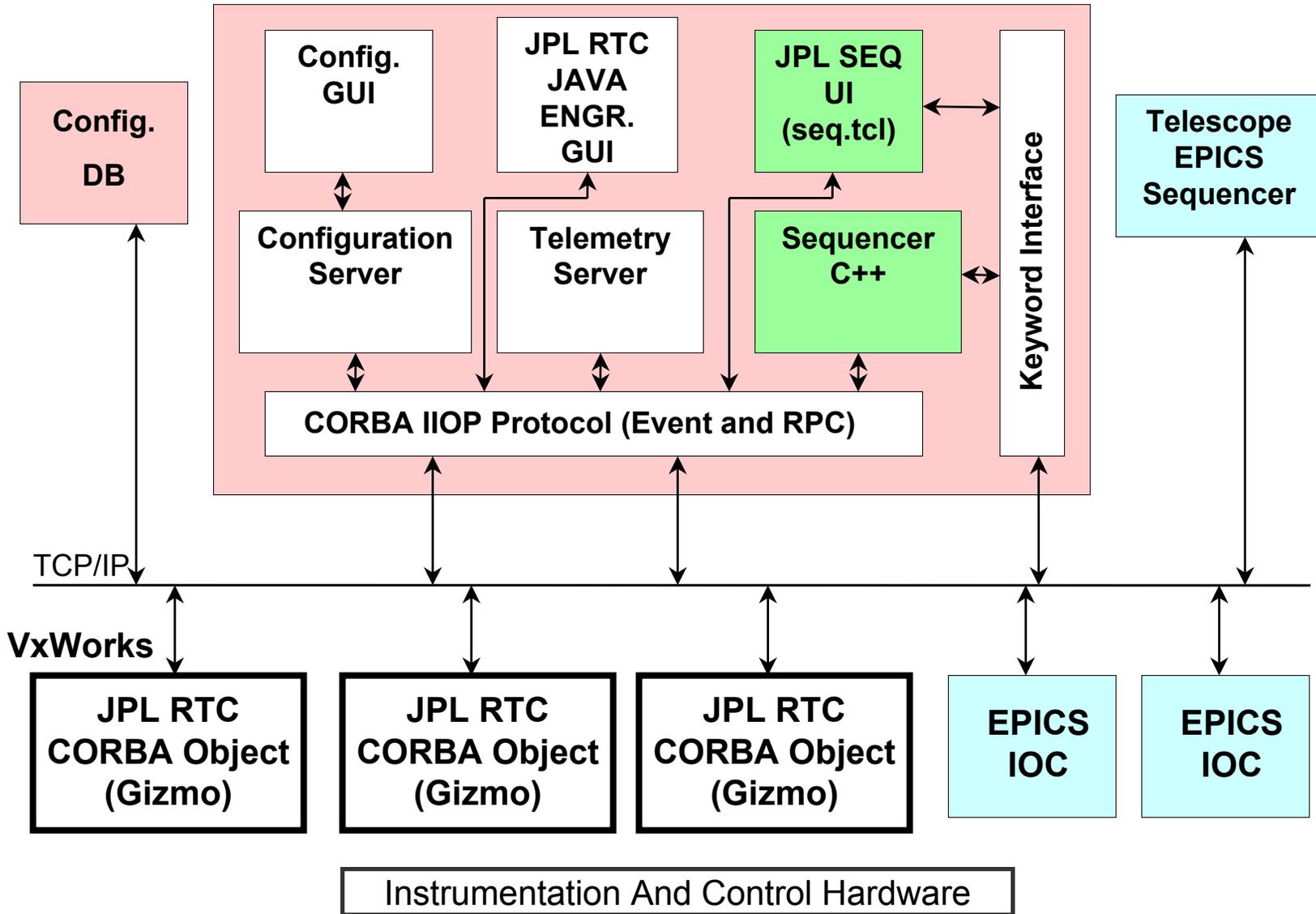
# Interferometer Sequencer Development

- Normal operation requires intricate and repeatable sequential control of top-level events (e.g. sequencing).
- History
  - Initial first fringes supported via UNIX EPICS sequencer implemented in State Notation Language and C++.
  - Next generation science sequencer made using Rhapsody CASE tool.
- All implementations had Tcl/Tk UI (seq.tcl)
  - **Initially:** Keyword only.
  - **Next:** Addition of CORBA event service to consume (monitor) RTC-style telemetry.
  - **Now:** Fully combined CORBA/Keyword functionality.

# RTC-Toolkit and Sequencer



## WORKSTATION



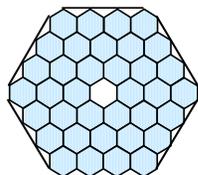
KECK INTERFEROMETER  
PROJECT



28 August 2002

Leonard.J.Redder@jpl.nasa.gov

# Interferometer Sequencer User Interface (seq.tcl)



KECK INTERFEROMETER  
PROJECT

Angle Tracker Telemetry

Fringe Tracker Telemetry

Selected observation record

Current observation & baseline vector

Sequencer observing controls

Text Status

Seq Revision: 1.25 (Rhapsody CORBA Experimental Version)

File Seq User

Keck Angle Tracker:

KAT 1 ServoMode:	SM_IDLE	TrackMode:	TCL_SEARCH	Centroid X:	5.5714717894	Y:	-9.169376796
KAT 2 ServoMode:	SM_IDLE	TrackMode:	TCL_SEARCH	Centroid X:	6.2576525865	Y:	6.5126806537

Fringe Tracker and FDLs:

F.T. Op. Mode:	OM_SERVO	ServoMode:	SM_IDLE	TrackMode:	TML_SEARCH
DelayLine 0 OPD:	-0.1065261012	DelayLine 1 OPD:	0.02573123982		None

HDC179422	19	11	30,983	+26	44	09,137	0,037	-0,032	6,3	5,2	F5V	5,6	0,30+/-0,1	cal
HDC190993	20	06	53,408	+23	36	51,931	0,017	0,000	5,1	5,6	B3V	8,5	0,25+/-0,1	cal
HDC196724	20	38	31,339	+21	12	04,225	0,080	-0,002	4,8	4,8	A0V	5,3	0,38+/-0,1	cal
HDC202573	21	15	57,173	+25	26	01,812	-0,072	-0,037	7,0	5,4	G5V	4,4	0,21+/-0,3	cal
HDC208057	21	53	03,769	+25	55	30,503	0,010	0,000	5,1	5,7	B3V	3,3	0,24+/-0,1	cal
HDC212395	22	23	39,565	+20	50	53,627	0,359	-0,015	6,2	4,9	F7V	5,9	0,36+/-0,1	cal
HDC159222	17	32	00,993	+34	16	16,128	-0,290	0,063	6,5	4,9	G5V	5,8	0,40+/-0,1	cal
HDC165567	18	04	43,198	+40	05	03,057	0,033	0,024	6,5	5,2	F7V	5,2	0,28+/-0,1	cal
HDC178911	19	09	04,384	+34	36	01,624	0,057	0,195	6,7	5,3	G1V	1,9	0,28+/-0,2	cal
HDC184499	19	33	27,081	+33	12	06,714	-0,555	0,224	6,6	5,2	G0V	5,8	0,34+/-0,1	cal

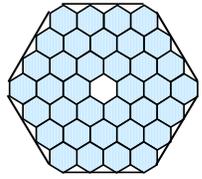
Observing:

Base Line East: 51.703989 North: 67.455287 Up: 0.001524 Offset: -1.41 DeltaUT: 0.0008333333333333

Baseline Comment: # Solution from Mar 01 with Nov 01 bias File: K1K2.baseline,v Active Subsystem: V2

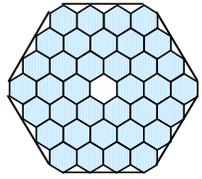
Simulation Mode  Stop  Off  On  Init  Prev  Next

Completed...  
HDC196724 20 38 31.339 +21 12 04.225 0.080 -0.002 4.8 4.8 A0V 5.3 0.38+/-0.1 cal is selected  
Opened observation baseline file:  
/net/kawela/local2/kroot.intdev/rel/if/qfix/1-5-3/baselines/K1K2.baseline  
# Best-fit model baseline (ENUBias) (m): 51.703989 67.455287 0.001524 -1.41  
# Baseline vector configuration: K1(fd10)->K2(fd11)  
Completed...  
Set all sequencer object subsystems to simulate mode.  
HDC208057 21 53 03.769 +25 55 30.503 0.010 0.000 5.1 5.7 B3V 3.3 0.24+/-0.1 cal is selected



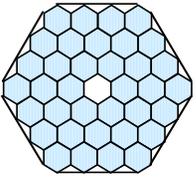
## Introduction to CORBA

- Keyword interface is procedural and defined by Keyword variable names that are set to values.
- **CORBA is object-oriented in that remote objects are defined and remote methods executed.**
  - **Objects are servers. Examples:**
    - RTC objects
    - Sequencer state machine objects
  - **Distributed framework standard**
    - Object Request Broker (ORB) used to mediate interaction between client and server.
    - ORB to ORB communication via Internet Inter-ORB Protocol (IIOP).
    - Interface Definition Language (IDL) used to define object structure (e.g. methods and variables).
    - Name Service resolves object reference from text name.



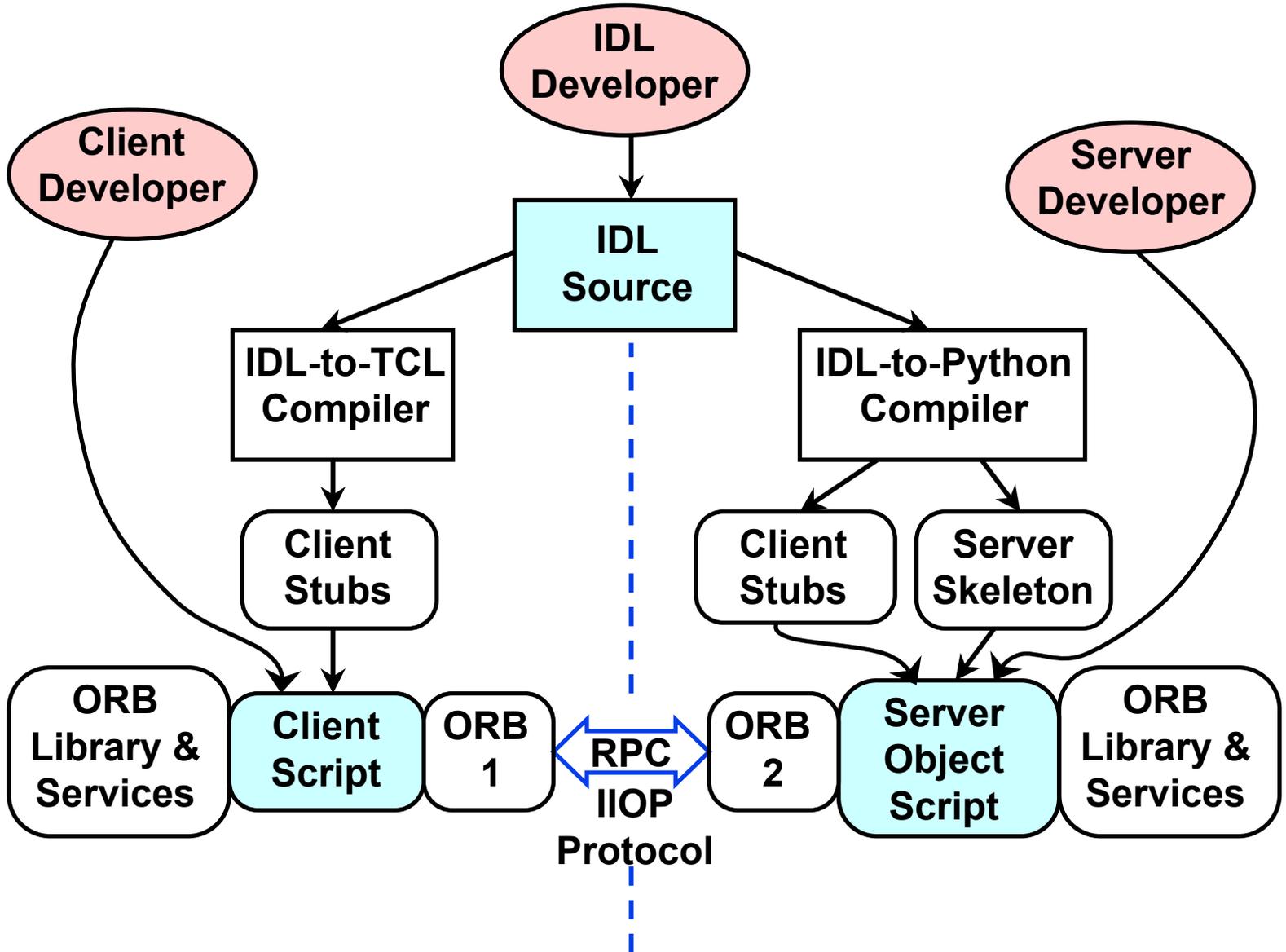
## CORBA Language ORBs

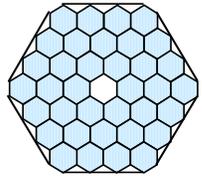
- C++            TAO, Mico, omniORB
- Tcl            Mico, Combat (incrTcl required)
- Python        omniORB, omniORBpy
- Perl            ORBit, ORBit-perl
- JAVA          JacORB
- Lisp            Allegro Common Lisp, ORBLink



# Client and Server CORBA Development Example

KECK INTERFEROMETER  
PROJECT



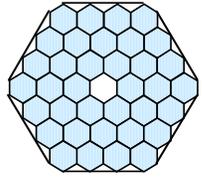


## Interface Definition Language (IDL) Example of TrackerModule Object

```
module TrackerModule {
  interface Tracker {
    exception InternalError {};

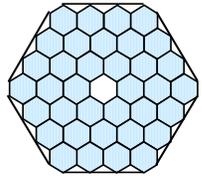
    // Brings the object to a stop and leaves it idle.
    void Idle() raises (InternalError);

    // Initiates tracking behavior for this object.
    void Track() raises (InternalError);
  };
}
```



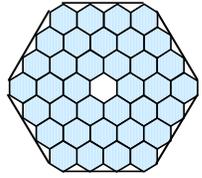
# The Components of a Client

- Initialize ORB.
- Define the objects.
  - Done at compile-time for C++ and other strongly-typed compiled languages.
  - Done at runtime for interpreted weakly-typed languages.
- Find the name service.
- Resolve the server object name.
- Invoke method(s).



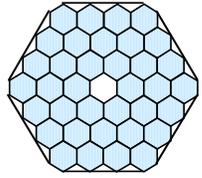
## Tcl Client Script Example for TrackerModule Object

```
#!/usr/bin/env combatsh
# A simple Tracker object client (client.tcl).
# Initialize CORBA ORB and set reference
# to name service on command line.
set argv [eval corba::init $argv]
# Source the interface repository into the Tcl shell.
source CosNaming.tcl
combat::ir add $_ir_CosNaming
source Tracker.tcl
combat::ir add $_ir_Tracker
# Obtain naming service IOR
set n [corba::resolve_initial_references NameService]
# Resolve object handle from naming service
set t [$n resolve {{id "spie" kind ""} {id "tracker" kind ""}}]
# Call the Track and Idle method on the object spie.tracker
# that was resolved from the name service.
$t Track
after 2000
$t Idle
```



# The Components of a Server

- Define Server Methods.
- Initialize ORB.
- Create the server object(s).
- Find the name service.
- Register the object(s) with name service.
- Run the ORB / Wait for shutdown.



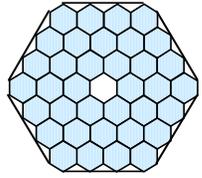
# A Python Example Implementation of TrackerModule Server (Part I)

```
#!/usr/bin/env python
# Simple TrackerModule server

import sys
from omniORB import CORBA
import CosNaming
from RTC import CorbaUtils
import TrackerModule, TrackerModule__POA

class TrackerObject(TrackerModule__POA.Tracker):
    def Idle(self):
        print "Tracker is now idling"

    def Track(self):
        print "Tracker is now tracking"
```



## A Python Example Implementation of TrackerModule Server (Part II)

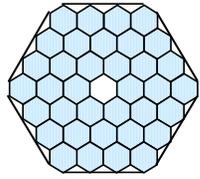
```
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
obj = orb.resolve_initial_references("NameService")
ns = obj._narrow(CosNaming.NamingContext)

poa = orb.resolve_initial_references("RootPOA")
poa._get_the_POAManager().activate()

tr = TrackerObject()
ref = tr._this()

name = CorbaUtils.string_to_name("spie.tracker")
CorbaUtils.create_reference(ns, name, ref)

orb.run()
```

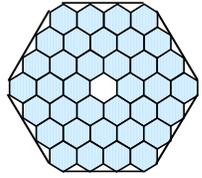


## Summary of Implementation

- Keck Interferometer Project (Mostly Keyword)
  - Alignment UI (aligner.tcl)
  - Sequencer UI (seq.tcl)
  - Various small unit test scripts.
  - Keck has several key telescope UIs.
- JPL Testbed Technology Demonstration (Various small Python and Perl scripts).
  - Configuration database interface have been created in both Python and Perl.
  - Delay line home position commanding (dlhome.py).
  - A Corba ping program in Python for identification of live RTC CORBA objects.
  - Perl implementation of CORBA server object for equipment power control.

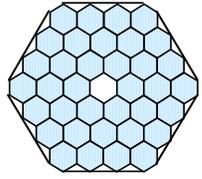
KECK INTERFEROMETER  
PROJECT





## Conclusion

- Demonstrated heterogeneous Keyword/CORBA functionality.
- Using KTcl and Combat for rapid development of integrated Keyword and CORBA functionality within single Tcl script is practical and easy!
- Tcl, Python, Perl and C++ (JAVA, Lisp) client and server programs can interoperate using CORBA standard middleware technology.
- Scripting language use is great for proof of concept and evolutionary design processes.
- Python is well suited for CORBA since it was designed from the start as an OO language and has a CORBA standard language mapping.



**KECK INTERFEROMETER  
PROJECT**

**BACKUP SLIDES**

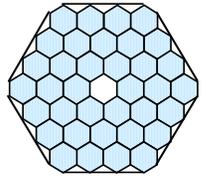


28 August 2002

[Leonard.J.Redder@jpl.nasa.gov](mailto:Leonard.J.Redder@jpl.nasa.gov)

# Python Client Script Example for TrackerModule Object.

```
#!/usr/bin/env python
# A Simple TrackerModule client
import sys, time
from omniORB import CORBA
import CosNaming
import TrackerModule
# Initialize CORBA ORB and set reference
# to naming service on command line.
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
# Obtain naming service IOR
obj = orb.resolve_initial_references("NameService")
ns = obj._narrow(CosNaming.NamingContext)
# Resolve object handle from naming service
name = [CosNaming.NameComponent(id = "spie", kind = "")]
name += [CosNaming.NameComponent(id = "tracker", kind = "")]
obj = ns.resolve(name)
tr = obj._narrow(TrackerModule.Tracker)
# Call the Track and Idle method on the object spie.tracker
# that was resolved from name service.
tr.Track()
time.sleep(2)
tr.Idle()
```



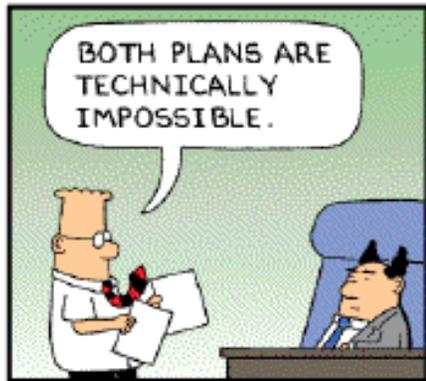
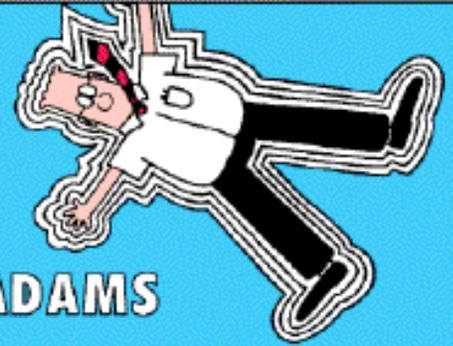
## Perl Client Script Example for TrackerModule Object

```
#!/usr/bin/env perl
# A Simple TrackerModule client
# ./client.pl -ORBNamingIOR=IOR:...
use CORBA::ORBit idl => [ qw(/usr/include/orbsvcs/CosNaming.idl) ];
use CORBA::ORBit idl => [ qw(TrackerModule.idl) ];
use strict;
# Ensure that we enable TCP/IP communications for the ORB
unshift @ARGV, "-ORBIIOPv4=1";
# Initialize CORBA ORB and set reference
# to naming service on command line.
my $orb = CORBA::ORB_init(@ARGV);
# Obtain naming service IOR
my $ns = $orb->resolve_initial_references("NameService");
my $nsior = $orb->object_to_string($ns);
# Resolve object handle from naming service
my $name = [{'id' => 'spie', 'kind' => ''}, {'id' => 'tracker',
'kind' => ''}];
my $Tracker = $ns->resolve($name);
# Call the Track and Idle method on the object spie.tracker
# that was resolved from name service.
$Tracker->Track;
sleep 2;
$Tracker->Idle;
```



# DILBERT®

BY  
SCOTT ADAMS





Copyright © 2002 United Feature Syndicate, Inc.